# Chapitre 1
# Représentation des nombres sur ordinateur

ENSAM-Meknès

Université Moulay Ismail - Meknès

28 septembre 2011

L'analyse Numérique est une spécialité récente des mathématiques, elle s'est développée avec l'apparition des ordinateurs.
Son objet essentiel est d'écrire les programmes permettant d'effectuer les opérations nécessaires pour résoudre numériquement un problème donné.

Les méthodes ont un "coût", lié d'une part au temps de calcul, ie au nombre d'opérations élémentaires (additions, soustractions, multiplications et divisions) à faire, et d'autre part à l'espace mémoire nécessaire pour stocker les données et les résultats.

L'ordinateur ne fait pas de calculs exacts (à cause du mode de représentation des nombres). Ceci constitue un inconvénient car cela provoque des erreurs d'arrondis et de troncature.

Pour un grand nombre de problèmes, le seul moyen de calculer la solution, si elle existe, est de l'approcher numériquement.

## Un algorithme c'est

- une démarche pour obtenir une solution (impliquant des simplifications, ajout/retrait d'hypothèses sur le modèle, etc)
- ET l'implementation sur ordinateur de cette démarche.

## Objectifs de l'analyse numérique :

- proposer et développer des algorithmes (méthodes d'approximation) pour calculer une solution approchée,
- contrôler les diverses sources d'erreurs, propres à l'approximation numérique,
- évaluer la pertinence et la valeur de différents algorithmes (estimer leurs performances) pouvant être utilisés pour résoudre le même problème afin de sélectionner les meilleurs .

## Quelques problèmes typiques

- Interpolation - Extrapolation.
- Résolution de systèmes linéaires.
- Résolution de systèmes non-linéaires.
- Intégration et différentiation numérique.
- Résolution de systèmes d'équations différentielles.
- Résolution de systèmes d'EDP.
- Analyse statistique.
- Transformée de Fourier.
- Diagonalisation.
- Calcul de valeurs propres. item Calcul de vecteurs propres.
- ...

## Langages de programmation

- Fortran : Idéal pour le calcul intensif.
- C : le plus proche du "hardware".
- IDL : Courant en astrophysique.
- Java : Indépendant de la machine (portabilité).
- Shell : Scripting de base.
- Autre : C++, python, Mathematica, etc . . . .

A choisir en fonction de la tache à réaliser, et non en fonction des habitudes.

## Qu'est-ce qu'un bon algorithme ?

Afin de choisir le meilleur algorithme possible, il faut pouvoir les comparer. Un bon algorithme est :

- le moins coûteux possible en place mémoire,
- le moins coûteux possible en temps de calcul : c'est-à-dire qui minimise le nombre d'opérations nécessaires. C'est ce qu'on appelle un problème de complexité.
- le plus stable possible, ie le moins sensible aux erreurs d'arrondi que nous venons d'évoquer,
- le plus précis possible : la solution approchée obtenue est-elle proche ou loin de la solution exacte. C'est ce qu'on appelle l'estimation d'erreur.

## Exemple : Calcul du déterminant d'une matrice $N \times N$

La méthode mathématique de Cramer nécessite $N \times N!$ opérations élémentaires ($N!$ additions et $(N-1) \times N!$ multiplications).
Prenons N = 50, sachant que $50 \times 50! = 15.10^{55}$, et qu'un ordinateur peut faire environ 20 GFlops (ie 20 milliards d'opérations à la seconde), il faudrait quand même environ $10^{50}$ ans pour faire ce calcul.

## Les quatre principales sources d'erreurs :

- erreurs de modélisation,
- erreurs sur les données,
- erreurs dues à la représentation des nombres sur ordinateur,
- erreurs de troncature ou de discrétisation.

## Exemple 1

Soit la suite définie par : $x_{n+1} = 4x_n(1 - x_n)$ On peut la programmer de deux façons :

- $x_1 = 4 * x_1 * (1.0 - x_1)$ ;
- $x_2 = 4 * x_2 - 4 * x_2 * x_2$ ;

"Normalement", on doit trouver la même chose "mathématiquement", mais les opérations ne sont pas faites dans le même ordre en machine.

## Exemple 2

Soit à résoudre l'équation : $x^2 + 2bx - 1 = 0$ Où b peut être "très grand". On demande de calculer la solution positive : $x_+ = -b + \sqrt{b^2 + 1}$ Comment faut-il la programmer ? La aussi, deux possibilités :

- $x = -b + \sqrt{b * b + 1.0}$ ;
- $x = \frac{1.0}{b + \sqrt{b * b + 1.}}$

Équivalentes mathématiquement.

## Exemple 3

Soit à calculer, pour n fixé : $S_n = \sum_{i=1}^{n} \frac{1}{i}$ Il y a (au moins) deux façons d'écrire la boucle :

- for (i=1 ; i<=n ; i++) x = x + 1.0/i ;
- for (i=n ; i>=1 ; i–) x = x + 1.0/i ;

Sans oublier d'initialiser x à 0 évidemment. Ici aussi les mathématiques sont équivalentes.

## La modélisation c'est :

- L'identification des différents facteurs agissant dans le phénomène (physique, biologique, économique , etc) à l'étude.
- L'identification et réprésentation mathématique des relations entre les facteurs : écriture des "formules/équations" représentants ces relations.

## Les erreurs de modélisation peuvent provenir de

- l'étape de "mathématisation" i.e. simplification ou mauvaise description de la relation entre différents facteurs lors de la mise en équations
- pour réduire le degré de complexité d'un phénomème physique, on est souvent amené à simplifier le système d'équations, ce qui revient à négliger certaines composantes de la réalité physique.

Solution : Un choix convenable du modèle mathématique.

## Les erreurs sur les données proviennent généralement de

- l'imprécision des mesures physiques qui sont utilisées dans l'algorithme.

Solution : Ces erreurs peuvent généralement être réduites en améliorant la précision des mesures.

## Remarque

Il est possible d'étudier l'influence de ces erreurs sur le résultat final, par exemple à l'aide de la notion de conditionnement (sensibilité aux erreurs).

Dans la suite de ce cours, nous ne nous intéresserons pas aux erreurs de modélisation ni à celles sur les données.

## Définition 1

Soit $x$ un nombre réel et $\tilde{x}$ une approximation de $x$.
L'erreur absolue : $\Delta x = |x - \tilde{x}|$ (mesure de l'erreur)
L'erreur relative : $\dfrac{\Delta x}{|x|}$ (importance de l'erreur).

## Définition 2

Si $m$ est le plus petit entier signé tel que $\Delta x \leq 0.5 \times 10^m$ alors le chiffre correspondant à la $m^{\text{ième}}$ puissance de 10 (et tous ceux à sa gauche jusqu'au dernier non-nul) sont dit significatifs.

## Exemple

Si $x = \pi$ et $\tilde{x} = 3.1428$, d'où $\Delta x = 0.12\ 10^{-2} < 0.5\ 10^{-2}$. Donc le chiffre des centièmes est significatif, soit le chiffre 4 dans 3.1428 et on a en tout 3 chiffres significatifs (3.14).

- L'ordinateur doit représenter les nombres dans un système qui lui permette d'exécuter les opérations souhaitées efficacement.
- La structure interne des ordinateurs s'appuie généralement sur le système binaire : représentation des nombres en base 2.
- Les calculs sont généralement effectués en base 2, et les interactions avec l'usager affichés en base 10.

## Décimal $\rightleftarrows$ binaire

$$
\begin{aligned}
(100)_{10} &= 1 \times 10^2 + 0 \times 10^1 + 0 \times 10^0 \\
&= 1 \times 64 + 1 \times 32 + 0 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 0 \times 1 \\
&= 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 \\
&= (1100100)_2
\end{aligned}
$$

$(0.3)_{10} = (0.010\,01100\,01100\,...)_2 \leftarrow$ fraction périodique en binaire !

Sur un ordinateur l'unité d'information de base ou bit prend la valeur 0 ou 1. On regroupe les bits en mots de longueur variable (8, 16, 32 ou 64 sont les plus courants). Les nombres réels sont représentés à l'aide de ces mots. La capacité mémoire est finie par construction. Il est donc nécessaire de représenter les nombres réels sous forme approchée. Cela entraîne un certain nombre d'erreurs.

Le fait d'utiliser un nombre limité de bits pour représenter un nombre réel a des conséquences importantes. Pour représenter les nombres réels on recourt généralement à

La troncature à $N$ chiffres : on retranche les chiffres à partir de la position $N+1$.

L'arrondi à $N$ chiffres : on ajoute 5 au $(N+1)^{\text{ième}}$ chiffre de la mantisse avant d'effectuer la troncature.

## Nombres Entiers

La première codification des nombres consiste à les coder de façon naturelle sur les mots connus au niveau de la machine, 8, 16, 32 ou 64 bits la limitation étant la valeur maximale que l'on peut stocker. On parle alors de représentation en champs fixe.

Par nombre naturel, on entend les nombres entiers, positifs ou nuls

- 1 octet pourra coder un nombre de $(0000\,0000)_2 \rightarrow (1111\,1111)_2$ $(0 \rightarrow 255)$
- 1 mot de 16 bits : de $0 \rightarrow 2^{16} - 1$ (65535)
- 1 mot de 32 bits : de $0 \rightarrow 2^{32} - 1$ (4 294 967 295)

La limite reste toujours la taille du mot manipulé

La représentation des nombres négatifs est réalisé grâce au bit de poids fort "bit de gauche". On parle également d'entiers signés.

- Nombres positifs $\rightarrow$ le bit de poids fort est 0
- Nombres négatifs $\rightarrow$ le bit de poids fort est 1

Les autres bits contiennent la valeur absolue du nombre

## Exemple sur un octet

| Décimal | Binaire | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 12 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| -12 | **1** | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

Inconvénient : Pour réaliser des opérations entre nombres signés, il faut faire un traitement particulier du bit de signe. Le résultat final ne pouvant être obtenu de façon simple (addition ou soustraction des deux nombres).
D'où l'utilisation du complément à 2.

## Représentation par le complément à 1

Le complément à 1 (noté C1) est également appelé complément logique, il consiste à inverser chaque bit ($0 \rightarrow 1$ et $1 \rightarrow 0$)

## Représentation par le complément à 2

Le complément à 2 (noté C2) est également appelé complément vrai, il consiste à ajouter 1 en binaire au complément à 1.

- L'entier positif est représenté en binaire naturel.
- L'entier négatif est représenté par le complément à 2 de son opposé.

| Décimal | Binaire | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 12 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| C1 | **1** | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| +1 | | | | | | | | 1 |
| -12 | **1** | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

## Exemple

Une soustraction $a - b$ devient une addition $a + (-b)$. $(-b)$ étant le complément à 2 de $b$

| Décimal | Binaire | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 17 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| -12 | **1** | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| | **0** | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

| Décimal | Binaire | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| -12 | **1** | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| | **1** | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

Si le bit de poids fort est 0, le résultat est positif, et si le bit de poids fort est 1, le résultat est négatif Pour connaître la valeur absolue du résultat négatif, il suffit de faire le complément à 2 :

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **1** | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| C1 | **0** | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| +1 | | | | | | | | 1 |
| | **0** | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

En effet $|5 - 12| = (0000\,000111)_2 = (7)_{10}$

## Les limites des nombres

Les nombres sont codés de façon fixe, ce qui fait qu'ils ont une limite liée au nombre de bis qui les représentent :

| Nombres de bits | + grand nombre positif | + petit nombre négatif |
|---|---|---|
| 8 | **0**111 1111 → +127 | **1**000 0000 → -128 |
| 16 | **0**111 1111 1111 1111 → 32767 | **1**000 0000 0000 0000 →- 32768 |
| 32 | 4 294 967 295 | - 4 294 967 296 |

En synthèse, si le nombre de bits est $n$, les valeurs vont de $-2^n$ à $2^n - 1$

On se donne un entier $b \geq 2$ appelé base de numération. On choisit aussi un entier $p \geq 2$ appelé précision ou nombre de chiffres significatif. On se donne aussi deux entiers $e_{max} \geq 2$ et $e_{min} \leq -2$. Un nombre flottant normalisé est alors un nombre de la forme

$$x = (-1)^s \times m \times b^e$$

où $s \in \{0, 1\}$ et $(-1)^s$ est le signe de $x$. $m$ est appelée La mantisse, elle est écrite sous la forme d'un nombre avec virgule fixe et possédant un nombre maximum de $p$ chiffres significatifs

$$m = 0.d_1 d_2 ... d_p = \sum_{k=1}^{p} d_k \; b^{-k},$$

avec $1 \leq d_1 \leq b - 1$ (mantisse normalisée) et $0 \leq d_k \leq b - 1$. L'entier $e$, appelé exposant , vérifie $e_{min} \leq e \leq e_{max}$.

On vérifie que $\frac{1}{b} \leq m \leq 1 - b^{-p}$ et
$\lambda = b^{e_{min}-1} \leq |x| \leq \mu = (1 - b^{-p}) b^{e_{max}}$. L'ensemble des flottants normalisés est noté $F(b, p, e_{min}, e_{max})$

Par exemple, prenons b = 10, p = 5 et $e_{max} = -e_{min} = 100$.
Le rationnel 0.000000000000000000000123 est dans $F(10, 5, -100, 100)$ et s'écrit

$$x = (-1)^0 \times 1.2300 \times 10^{-22}$$

. Notons que 0 n'est pas un entier flottant normalisé.

On pose $\epsilon = b^{1-p}$ et on vérifie sans peine le lemme suivant.

Lemme 1 Soit $x > 0$ un entier flottant normalisé et $y$ le suivant (le plus petit entier flottant normalisé plus grand que $x$) s'il existe. Alors $y - x$ vérifie

$$\frac{\epsilon|x|}{b} \leq y - x \leq \epsilon x$$

Une fois choisis $b$, $p$, $e_{min}$ et $e_{max}$ on se donne une application d'arrondi

$$\phi : \mathbf{R} \to F(b,p,e_{min},e_{max}) \bigcup \{OVERFLOW, UNDERFLOW\}$$

jouissant des propriétés suivantes :

- $\phi(x) = OVERFLOW$ si et seulement si $|x| > \mu$ ;
- $\phi(x) = UNDERFLOW$ si et seulement si $|x| < \lambda$ ;
- $\phi$ est croissante sur $[\lambda, \mu]$ et sur $[-\mu, -\lambda]$ ;
- La restriction de *phi* à $F(b,p,e_{min},e_{max})$ est l'identité.

Le fait d'utiliser un nombre limité de bits pour représenter un nombre réel a des conséquences importantes :

- Cette représentation introduit une erreur (troncature ou arrondi) qui peut avoir un impact important sur la précision des résultats.
- Quelque soit le nombre de bits utilisés, il existe un plus petit et un plus grand nombre (positif et négatif) représentables . Il existe donc un intervalle fini hors duquel on a un débordement (overflow) ou un sous-dépassement (underflow). Dans ce cas les operations arithmétique n'ont plus de sens.
- A l'intérieur de cet intervalle fini, seul un nombre fini de nombres sont représentables exactement.

La représentation en virgule flottante induit une erreur relative qui dépend du nombre de bits de la mantisse, de l'utilisation de la troncature ou de l'arrondi, ainsi que du nombre $x$ à représenter.

## Définition

La précision machine est la plus grande erreur relative commise en représentant un nombre réel sur ordinateur :

$$\text{precision machine} = \text{maximum} \frac{\Delta x}{|x|}$$

En utilisant la troncature, on a : $\varepsilon_m = b^{1-N}$
En utilisant l'arrondi, on a : $\varepsilon = \varepsilon_m/2 = b^{1-N}/2$.

Remarque : En réalité la précision machine est au maximum égale à $\varepsilon_m$ ou $\varepsilon$ mais dans la pratique on confond la précision avec cette borne.

## La convention IEEE-754

Norme visant à standardiser la représentation des nombres sur les ordinateurs. En particulier elle impose le nombre de bits pour la mantisse, et l'exposant pour les nombres reels à 32 bits (simple precision) et les reels 64 bits (double precision). Fait en sorte que l'on peut determiner la precision machine ainsi que le plus grand et plus petit reels (simple et double precision) représentable indépendemment de l'ordinateur .

| | simple precision | double precision |
|---|---|---|
| $\varepsilon_m$ | $0.119 \times 10^{-6}$ | $0.222 \times 10^{-15}$ |
| plus petit réel (normalisée) | $1.2 \times 10^{-38}$ | $2.2 \times 10^{-308}$ |
| plus petit réel (non-normalisée) | $1.4 \times 10^{-45}$ | $4.9 \times 10^{-324}$ |
| plus grand réel | $3.4 \times 10^{38}$ | $1.8 \times 10^{308}$ |

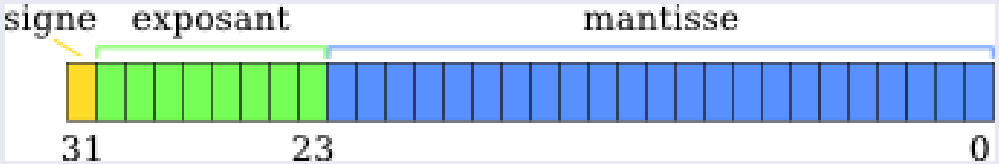
## Décalage de l'exposant

L'exposant étant représenté par un nombre signé (complément à 2), la comparaison entre les nombres flottants devient un peu plus difficile. Pour remedier à ce problème, l'exposant est décalé, afin de le stocker sous forme d'un nombre non signé. Ce décalage est de $2^{e-1} - 1$ ($e$ représente le nombre de bits de l'exposant) ; il s'agit donc d'une valeur constante une fois que le nombre de bits $e$ est fixé :

$$valeur = signe \times 1, mantisse \times 2^{(exposant - decalage)}$$

## Format simple précision (32 bits)

Un nombre flottant simple précision est stocké dans un mot de 32 bit : 1 bit de signe, 8 bits pour l'exposant et 23 pour la mantisse. L'exposant est donc décalé de $2^{8-1} - 1 = 127$ dans ce cas. L'exposant -127 (qui est décalé vers la valeur 0) est réservé pour zéro et les nombres dénormalisés, tandis que l'exposant 128 (décalé vers 255) est réservé pour coder les infinis et les NaNs.



## Exemple1

Pour 0 01111100 01000000000000000000000, l'exposant est $124 - 127 = -3$ et la partie significative est 1,01 soit 1,25 en décimal $(1x2^0 + 0x2^{-1} + 1x2^{-2})$ ; le nombre représenté est donc $+1,25x2^{-3}$ soit $+0,15625$.

## Exemple2

Codons le nombre -118,625 en utilisant IEEE 754.

1. C'est un nombre négatif, le signe est donc "1".
2. Nous écrivons le nombre (sans le signe) en binaire. Nous obtenons 1110110,101.
3. Nous décalons la virgule vers la gauche, de façon à ne laisser qu'un 1 sur sa gauche : 1110110,101 (bin) = 1,110110101 (bin) x $2^6$. C'est un nombre flottant normalisé : la mantisse est la partie à droite de la virgule, remplie de 0 vers la droite pour obtenir 23 bits. Cela donne 110 1101 0100 0000 0000 0000 (on omet le 1 avant la virgule, qui est implicite).
4. L'exposant est égal à 6, et nous devons le convertir en binaire et le décaler. Pour le format 32-bit IEEE 754, le décalage est $2^{8-1} - 1 = 127$. Donc 6 + 127 = 133 (dec) = 1000 0101 (bin).

-118,625 (dec) = 1100 0010 1110 1101 0100 0000 0000 0000 (bin) = C2ED4000 (hexa)

## Format double précision (64 bits)

Le format double précision est identique au simple précision, mis à part le fait que les champs sont plus grands. En effet, il possède 52 bits de mantisse, et 11 bits d'exposant. La mantisse est très élargie, alors que l'exposant est peu élargi. Ceci est dû au fait que, selon les créateurs du standard, la précision est plus importante que l'amplitude. Les NaNs et les infinis sont représentés en mettant tous les bits de l'exposant à 1 (2047). Pour les nombres normalisés, le décalage de l'exposant est +1023.

Les opérations élémentaires sont effectuées en arithmétique flottante de la manière suivante :

$$\begin{array}{rcl} x + y & \longrightarrow & fl(\, fl(x) + fl(y)\,) \\ x - y & \longrightarrow & fl(\, fl(x) - fl(y)\,) \\ x \div y & \longrightarrow & fl(\, fl(x) \div fl(y)\,) \\ x \times y & \longrightarrow & fl(\, fl(x) \times fl(y)\,) \end{array}$$

Attention : plusieurs propriétés de l'arithmétique : associativité, distributivité, etc ne sont plus valides (pas toujours) en arithmétique flottante !

Par exemple : supposons que les réels soient calculés avec $n = 3$ chiffres significatifs et arrondis à la décimale la plus proche.
Soient $x = 8,22, \quad y = 0,00317, \quad z = 0,00432$
$(x + y) + z \neq x + (y + z)$

Encore avec n =3

$$x \times (y + z) \neq x \times y + x \times z$$

$$\begin{aligned}
122 \times (333 + 695) &= fl(\, 0,122\ 10^3 \times fl(\, 0,333\ 10^3 + 0,695\ 10^3 \,)\,) \\
&= fl(\, 0,122\ 10^3 \times fl(\, 1,028\ 10^3 \,)) \\
&= fl(\, 0,122\ 10^3 \times fl(\, 0,103\ 10^4 \,)) \\
&= fl(\, 0,012566\ 10^7 \,)) \\
&= 0,126\ 10^6
\end{aligned}$$

$$\begin{aligned}
(122 \times 333) + (122 \times 695) &= fl(\, fl(\, 0,122\ 10^3 \times 0,333\ 10^3 \,) \\
&\quad + \ fl(\, 0,122\ 10^3 \times 0,695\ 10^3 \,)\,) \\
&= fl(\, fl(\, 0,040626\ 10^6 + fl(\, 0,08479\ 10^6 \,) \\
&= fl(\, 0,406\ 10^5 + 0,848\ 10^5 \,) \\
&= fl(\, 1,254\ 10^5 \,) \\
&= 0,125\ 10^6
\end{aligned}$$

Il ya donc une différence entre les deux résultats.

Il faut être prudent avec l'addition et la soustraction. Lorsque les exposants ne sont pas les mêmes il est nécessaire de décaler la mantisse avant d'effectuer l'addition ou la soustraction.

Pour $n = 4$

$$\begin{aligned} 0,4035\ 10^6 + 0,1978\ 10^4 &= fl(\,0,4035\ 10^6 + 0,1978\ 10^4\,) \\ &= fl(\,0,4035\ 10^6 + 0,001978\ 10^6\,) \\ &= fl(\,0,405478\ 10^6\,) \\ &= 0,4055\ 10^6 \end{aligned}$$

et

$$\begin{aligned} 0,56789\ 10^4 - 0,1234321\ 10^6 &= fl(\,0,5679\ 10^4 - 0,1234\ 10^6\,) \\ &= fl(\,0,005679\ 10^6 - 0,1234\ 10^6\,) \\ &= fl(\,-0,11772\ 10^6\,) \\ &= -0,1177\ 10^6 \end{aligned}$$

Il existe d'autre part, un certain nombre d'opérations risquées.
Soustraire deux nombres presque identiques :

$$\begin{aligned}\sqrt{7001}-\sqrt{7000} &= 0,8367\ 10^2 - 0,8367\ 10^2\\ &= 0,000\ 10^0 \quad \text{si } n=4\end{aligned}$$

$$\begin{aligned}\sqrt{7001}-\sqrt{7000} &= 0,83672\ 10^2 - 0,83666\ 10^2\\ &= 0,6\ 10^{-2} \quad \text{si } n=5\end{aligned}$$

Dans ce cas-ci on peut remédier à cette difficulté en évitant la soustraction des racines :

$$\sqrt{x}-\sqrt{y}=\frac{x-y}{\sqrt{x}+\sqrt{y}}$$

et on trouve $0,5977\ 10^{-2}$ au lieu de 0 pour $n=4$.

L'ordre dans lequel on fait les opérations et le nombre d'opérations ont un impact important sur la précision du résultats. Ainsi on devrait "toujours" :

- éviter les opérations avec des nombres très différents en ordre de grandeurs,
- éviter si possibles les soustractions avec des nombre relativement proche,
- on devrait faire les sommes en ordre décroissant.